VAŽNA NAPOMENA:

- Ova pitanja NISU materijal za pripremu kolokvijuma, već materijal za proveru znanja i vežbanje zadataka.
- Za pripremu kolokvijuma koristite PDF materijale koji su raspoloživi na stranici predmeta.

1.1. Koji su osnovni ciljevi koje je neophodno da operativni sistem postigne pri posredovanju između korisnika i računarskog sistema?

Operativni sistem treba da:

- izvršava korisničke programe i olakša rešavanje korisničkih problema,
- korišćenje računarskog sistema učini podesnijim za korisnika i
- omogući efikasnije korišćenje hardvera računarskog sistema.

1.2. Navedite osnovne karakteristike operativnih sistema.

- Konkurentnost, odnosno postojanje više simultanih, paralelnih aktivnosti, kao što je koegzistencija više programa u memoriji,
- deoba resursa,
- postojanje dugotrajne memorije,
- determinizam po pitanju izvršavanja programa i nedeterminizam po pitanju opterećenja.

1.3. Definišite efikasnost i premašenje operativnog sistema.

Efikasnost e je odnos vremena u kom procesor radi za korisnika i ukupnog vremena potrebnog da se određeni posao ili grupa poslova obavi, tj: $e=t_{korisno}/t_{ukupno}$ $(0<e<1)$.

Premašenje o (*overhead*) je odnos vremena u kome procesor radi na održavanju samog sistema i ukupnog vremena: $o=t_{održavanje}/t_{ukupno}$ $(0<o<1)$.

1.4. Jedna od poželjnih osobina operativnog sistema je visoka efikasnost. U kom slučaju se može dozvoliti da operativni sistem "pregazi" ovaj princip i nepotrebno "pojede" resurse?

Na jednokorisnički sistemima iskorišćenje sistema uvećava se uvođenjem grafičkog korisničkog interfejsa. GUI dodatno opterećuje procesor i memoriju, ali optimizuje interakciju između korisnika i sistema i kao takav smatra se prihvatljivim gubitkom.

1.5. Zašto ne postoje višekorisnički jednoprocesni operativni sistemi?

Višekorisnički operativni sistemi obezbeđuju više virtuelnih mašina, tj. omogućavaju većem broju korisnika da koriste sistem istovremeno. Svaki korisnik u sistem donosi nove procese i zahteva njihovo izvršenje. U opštem slučaju više korisnika može zahtevati

izvršenje svojih procesa počev od istog trenutka. Takvi procesi se moraju izvršavati paralelno ili kvaziparalelno, što na jednoprocesnim operativnim sistemima nije moguće.

1.6. U čemu je razlika između simetričnog (SMP) i asimetričnog multiprocesiranja u višeprocesorskim sistemima?

SMP – svi procesori su ravnopravni (nema odnosna gospodar-rob) i svaki procesor izvršava istu kopiju operativnog sistema, pri čemu te kopije međusobno komuniciraju kad god je to potrebno. U idealnom slučaju, svakom procesoru se dodeljuje jedan proces koji se izvršava nezavisno od procesa na ostalim procesorima.

Asimetrično multiprocesiranje – svakom procesoru je dodeljen specifičan posao. Glavni procesor kontroliše ceo sistem i dodeljuje poslove ostalim procesorima.

1.7. Bazu podataka sa pratećom aplikacijom je potrebno postaviti na četiri servera. Kako se mogu udružiti ovi serveri ukoliko je potrebno obezbediti (a) visoku pouzdanost, (b) visoke performanse?

(a) Asimetrično – jedan čvor (server) opslužuje zahteve, dok su ostali u budnom, ali neaktivnom stanju. U slučaju otkaza glavnog servera, jedan od pratećih servera preuzima ulogu glavnog servera.

(b) Simetrično – svi serveri su aktivni i izvršavaju aplikaciju (bazu).

1.8. Koje funkcionalne grupe programa čine operativni sistem?

- Upravljanje procesorom,
- upravljanje memorijom,
- upravljanje ulazom i izlazom,
- upravljanje podacima (datotekama),
- upravljanje sekundarnom memorijom,
- umrežavanje,
- zaštita i
- korisnički interfejs.

1.9. a. Koja je osnovna karakteristika operativnih sistema sa mikrokernel arhitekturom?

b. Zašto su ovakvi sistemi pouzdaniji od monolitnih?

(a) Osnovna karakteristika operativnih sistema sa mikrokernel arhitekturom je postojanje minimalnog, pouzdanog jezgra visokih performansi. Sve druge funkcije jezgra realizuju se kao zasebni korisnički moduli koji se jednostavno mogu dodati bez uticaja na osnovno jezgro.

(b) Ovakav sistem radi pouzdanije jer se mnogo manje koda se izvršava u sistemskom režimu.

1.10. Objasnite kako razdvajanje korisničkog i sistemskog režima funkcioniše kao poseban oblik zaštite sistema.

Programi se u korisničkom režimu izvravaju na zahtev korisnika, a u sistemskom režimu na zahtev operativnog sistema. Privilegovane instrukcije mogu se izvršiti jedino u sistemskom režimu. Operativni sistem ima pristup privilegovanom setu instrukcija i pomoću njega uspostavlja kontrolu nad celim sistemom u svakom trenutku. Projektant operativnog sistema ne sme dozvoliti korisničkim programima da uspostave kontrolu nad računarom u korisničkom režimu – time se ostvaruje poseban oblik zaštite sistema.

1.11. Koja od sledećih instrukcija ne mora biti privilegovana: (a) postavljanje vrednosti tajmera, (b) očitavanje sistemskog časovnika, (c) brisanje memorije, (d) isključivanje prekida ili, (e) prelazak iz korisničkog u sistemski režim?

Instrukcija pomoću koje se očitava sistemski časovnik ne mora biti privilegovana.

1.12. Koja je uloga sistemskih poziva?

Sistemski pozivi omogućavaju procesima koji rade u korisničkom režimu da zatraže uslugu operativnog sistema. Na taj način korisnički program može da zatraži izvršenje privilegovanih instrukcija, kao što je rad sa ulazno-izlaznim uređajima.

2.1. Koji su hardverski preduslovi neophodni za realizaciju jezgra operativnog sistema, odnosno za nadogradnju hardvera jezgrom u hijerarhijskom modelu?

- Mehanizam prekida (omogućava izvršavanje kontrolnog programa, odnosno prebacivanje kontrole sa korisničkog programa na rutinu operativnog sistema),
- zaštitni mehanizam adresiranja memorije (sprečava pogrešno adresiranje, odnosno mogućnost da jedan proces svoje podatke upiše deo memorije dodeljen drugom procesu),
- privilegovani set instrukcija (omogućavaju operativnom sistemu da maskira prekide, dodeli procesor drugom procesu, pristupi zaštićenim memorijskim adresama ili obavi ulazno-izlaznu operaciju) i
- *real-time* časovnik (omogućava zakazivanje i raspoređivanje poslova).

2.2. Navedite osnovne delove jezgra prema modelu A. Lister-a.

- Prvi nivo obrade prekida (rutine za određivanje uzroka prekida i iniciranje odgovarajuće prekidne rutine),
- dispečer sistema, odnosno planer poslova niskog nivoa (dodeljuje procesor procesima) i
- rutine za ostvarivanje komunikacije između procesa.

2.3. Šta je proces i šta sve obuhvata?

Najjednostavnije rečeno, proces je program ili deo programa u stanju izvršavanja. Proces je aktivna dinamička celina, koji obuhvata:

- tri memorijske sekcije (programsku sekciju, u kojoj se nalazi kod, stek sekciju i sekciju podataka),
- vrednost programskog brojača i ostalih registara procesora koji su od interesa i
- ulazno-izlazne resurse koje eventualno koristi, kao što su, na primer, datoteke.

2.4. Šta je kontrolni blok procesa i šta je njime omogućeno?

Kontrolni blok procesa je deo operativne memorije, odnosno memorijska struktura koja sadrži kontrolne informacije neophodne za upravljanje tim procesom (jedinstveni identifikator procesa – PID, kontekst, prioritet, trenutno stanje i informacije o resursima koje proces koristi).

Pomoću kontrolnog bloka omogućeno je multiprogramiranje, odnosno višestruko prekidanje i nastavak izvršenja procesa.

2.5. Ukratko opišite stanja u osnovnom dijagramu stanja procesa (konačni automat sa pet stanja).

- START: nastanak procesa,
- READY: proces je spreman za rad, dobio je sve potrebne resurse osim procesora, i čeka da mu dispečer dodeli procesor,
- RUN: procesor izvršava instrukcije tekućeg procesa,
- WAIT: proces čeka na neki događaj jer su mu za dalje izvršenje potrebni neki resursi koji mu trenutno nisu na raspolaganju,
- STOP: kraj izvršenja procesa.

2.6. a. Definišite tranzicije RUN-WAIT i WAIT-READY.

b. Da li je na jednoprocesnim sistemima moguća tranzicija RUN-WAIT?

(a) RUN-WAIT: oduzimanje procesora procesu ukoliko je neki resurs neophodan za izvršenje procesa u međuvremenu postao zauzet ili nedostupan.

WAIT-READY: proces se vraća na kraj procesorskog reda posle oslobađanja resursa koji je neophodan za njegovo izvršenje.

(a) Tranzicija RUN-WAIT je moguća i u višeprocesnim i u jednoprocesnim operativnim sistemima. Na primer, u oba slučaja štampanje ne može da se nastavi dok se u štampač ne doda papir, s tim što se u slučaju višeprocesnog sistema u stanje WAIT dovodi print spooler, a u slučaju jednoprocesnog proces koji je inicirao štampanje.

2.7. a. Kada proces prelazi iz stanja RUN u stanje READY?

b. Kada je ova tranzicija moguća?

(a) Proces prelazi iz stanja RUN u stanje READY posle isteka vremenskog kvantuma.

(b) Ova tranzicija je moguća samo ako operativni sistem podržava pretpražnjenje.

2.8. Koji su mogući uzroci za suspendovanje procesa u proširenom dijagramu stanja procesa?

- Korisnik privremeno suspenduje proces da bi oslobodio resurse za izvršenje drugih procesa,
- operativni sistem suspenduje neke procese da bi sprečio pojavu zastoja i efekat zasićenja usled prevelike količine keširanih podataka.

2.9. Koje su osnovne funkcije (a) planera poslova i (b) dispečera sistama.

(a) Planer poslova deli poslove na procese, procesima dodeljuje prioritet na osnovu određenih algoritama i dovodi procese u red čekanja na procesor.

(b) Dispečer sistema odličuje koji će proces, kada i koliko dugo dobiti procesor. Prilikom zamene konteksta procesa, odnosno dodele procesora drugom procesu iz reda, dispečer pamti stanje procesa koji se prekida (kako bi se kasnije mogao nastaviti) i puni memoriju stanjem novog procesa kome se dodeljuje procesor.

2.10. Šta obavljaju sistemski pozivi fork i exec na UNIX sistemima?

Proces roditelj kreira novi proces pomoću fork sistemskog poziva. Sistemski poziv fork duplira kreira kopiju adresnog prostora roditelja i dodeljuje je detetu. Adresni prostor procesa deteta puni se programom koji treba da se izvrši pomoću sistemskog poziva *exec*.

2.11. Navedite primer u kojima višenitni procesi ne postižu bolje performanse od jednonitnih procesa.

Programi čiji se kod izvršava strogo sekvencijalno, odnosno ne može se izdeliti na manje podzadatke koje se mogu izvršavati paralelno.

2.12. U čemu je razlika između sinhronog i asinhronog slanja i primanja poruka?

Ukoliko je slanje poruka blokirajuće, proces koji šalje poruke se blokira, dok drugi proces kom je ta poruka namanjena ne primi poruku. Ukoliko je slanje ne-blokirajuće, proces koji šalje poruku nastavlja svoje aktivnosti ne čekajući potvrdu o prijemu. Ukoliko je primanje poruke blokirajuće, proces koji prima poruku se blokira sve dok poruku ne dobije. Ukoliko je primanje poruke neblokirajuće, proces će pokušati da primi poruku; ako je poruka stigla, prima je, a ako nije, kao rezultat se prihvata prazan niz, a proces nastavlja da radi dalje bez blokade.

2.13. U čemu je razlika između direktne i indirektne komunikacije?

Direktna komunikacija: između para procesa koji žele da komuniciraju automatski se uspostavlja link. Link se uspostavlja samo za dva procesa.

Indirektna komunikacija: link se uspostavlja između para procesa koji dele sanduče. Linku se mogu pridružiti više od dva procesa. Između para procesa može postojati više različitih linkova, pri čemu svaki link odgovara jednom sandučetu.

3.1. Algoritmom za raspoređivanje procesa određen je red izvršenja procesa koji čekaju na dodelu procesora. Ukoliko se na sistemu nalazi n procesa, koliko postoji različitih rasporeda za izvršenje procesa?

Postoji $n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 2 \cdot 1$ mogućih rasporeda.

3.2. U čemu je razlika između raspoređivanja sa pretpražnjenjem i raspoređivanja bez pretpražnjenja?

U slučaju raspoređivanja sa pretpražnjenjem, procesor se može oduzeti procesu koji nije završio svoje aktivnosti (na primer, ukoliko istekne vremenski kvantum za dodelu procesora). U slučaju raspoređivanja bez pretpražnjenja, to nije moguće.

3.3. a. Pod kojim okolnostima se procesor dodeljuje drugom procesu?

b. Koje od tih okolnosti zahtevaju pretpražnjenje?

(a) Procesor se dodeljuje drugom procesu ukoliko:

- tekući proces pređe u stanje čekanja resurs (na primer, čeka kraj ulazno-izlazne operacije koju je inicirao),
- proces roditelj čeka da proces dete završi svoje aktivnosti,
- tekući proces završi svoje aktivnosti,
- tekućem procesu istekne vremenski kvantum ili
- naiđe proces višeg prioriteta.

(b) Dodela procesora nakon isticanja vremenskog kvantuma i nailazak procesa višeg prioriteta.

3.4. Posmatrajte algortiram sa pretpražnjenjem zasnovan na dinamički izmenljivim prioritetima, takav da se viši prioritet označava većim brojevima prioriteta. Svim procesima se dodeljuje inicijalni prioritet 0 kada uđu u red čekanja na procesor. Ukoliko proces čeka na procesor (stanje READY), prioritet mu se menja stepenom d1, a ako se izvršava (stanje RUN), prioritet se menja stepenom d2. Algoritam se prilagođava izmenom parametara d1 i d2.

a. Koji se algoritam dobija ukoliko je $d2>d1>0$?

b. Koji se algoritam dobija ukoliko je $d1<d2<0$?

(a) FCFS (*First come, first served*)

(b) LIFO (*Last in, first out*)

3.5. Kako se rešava problem zakucavanja procesa niskog prioriteta ukoliko se raspoređivanje procesa obavlja na osnovu prioriteta procesa?

Procesima se povećava prioritet sa vremenom provedenim u redu čekanja na procesor. Rezultujući prioritet se formira na osnovu početnog prioriteta koji proces dobija kada uđe u red čekanja na procesor i vremena provedenog u redu.

3.6. U kakvoj se vezi (ukoliko veza uopšte postoji) nalaze algoritmi za raspoređivanje na osnovu prioriteta i SJF (*shortest job first*)?

Najkraći posao ima najviši prioritet.

3.7. Zašto je poželjno da algoritam za raspoređivanje procesa češće dodeljuje procesor procesima koji dominantno koriste U/I ulazno-izlazne uređaje (*I/O-bound*)? Da li se na takav način stvara mogućnost zakucavanja procesa koji dominantno koriste procesor (*CPU-bound*)?

Procesi koji dominantno koriste U/I koriste procesor u relativno kratkim intervalima. Ukoliko im se procesor odmah dodeli, ovi procesi će brzo preći u stanje čekanja na resurs i oslobodiće procesor, tako da mogućnost zakucavanja *CPU-bound* procesa praktično ne postoji.

3.8. Kako veličina vremenskog kvantuma utiče na performanse RR algoritma?

U slučaju velikih kvantuma RR konvergira ka FCFS algoritmu. U slučaju vrlo malih vremenskih kvantuma, svaki proces se izvršava brziom od manjom od 1/*n* brzine procesora (pri čemu je *n* broj procesa). Što su kvantumi manji, broj prebacivanja konteksta je veći, pa će samim tim i efektivna brzina izvršavanja procesa biti manja.

3.9. Dat je sledeći skup procesa čija su vremena izvršavanja na procesoru (*CPU-burst time*, izražena u milisekundama) i prioriteti dati u sledećoj tabeli:

Proces	Vreme izvršavanja	Prioritet
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Procesi su u sistem naišli u poretku P1, P2, P3, P4, P5, svi približno u trenutku t=0.

a. Nacrtati Gantove karte dodele procesora ukoliko se raspoređivanje vrši na osnovu sledećih algoritama: FCFS, SJF bez pretpražnjenja, raspoređivanje na osnovu prioriteta bez pretpražnjenja (manji broj znači veći prioritet) i RR sa kvantumom Q=1.

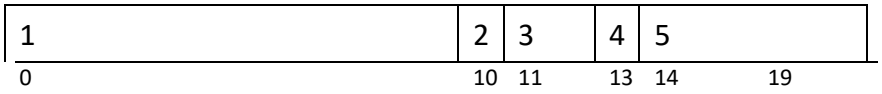
b. Odrediti vreme potrebno za kompletiranje procesa (*tournaround time*) za svaki proces (za sve gore pomenute algoritme).

c. Odrediti vreme čekanja za svaki proces i srednje vreme čekanja (za sve gore pomenute algoritme). Za koji je algoritam srednje vreme čekanja najmanje?

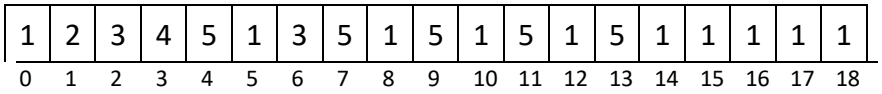
Kašnjenje dispečera zanemariti.

(a) Gantove karte dodele procesora:

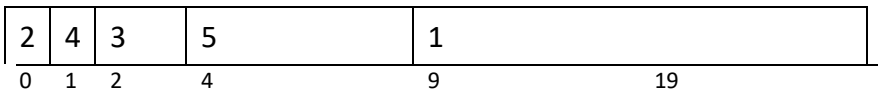
FCFS:



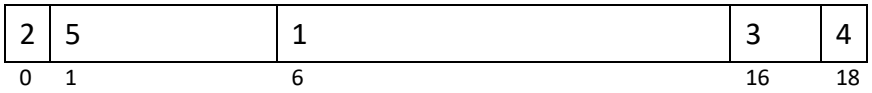
RR (Q = 1):



SJF (bez pretpražnjenja):



Rapoređivanje na osnovu prioriteta (bez pretpražnjenja):



(b) Vreme potrebno za kompletiranje procesa:

	FCFS	**RR**	**SJF**	**Prioritetno**
P1	10	19	19	16
P2	11	2	1	1
P3	13	7	4	18
P4	14	4	2	19
P5	19	14	9	6

(c) Vreme čekanja

	FCFS	**RR**	**SJF**	**Prioritetno**
P1	0	9	9	6
P2	10	1	0	0
P3	11	5	2	16
P4	13	3	1	18
P5	14	9	4	1
sr.vreme[1]	9.6	5.4	3.2	8.2

Srednje vreme čekanja je najmanje u slučaju SJF bez pretpražnjenja.

[1] Srednje vreme čekanja se računa kao srednja vrednost vremena čekanja za sve procese.

3.10. Dat je sledeći skup procesa čija su vremena nailaska u sistem (*arrival time*) i vremena izvršavanja na procesoru, izražena u milisekundama data u sledećoj tabeli:

Process	Vreme nailaska	Vreme izvršavanja
P1	0	8
P2	0.4	4
P3	1	1

Nacrtati Gantovu kartu i odrediti srednje vreme potrebno za kompletiranje procesa i srednje vreme čekanja[2] ukoliko se raspoređivanje procesa obavlja po:

a. FCFS algoritmu,

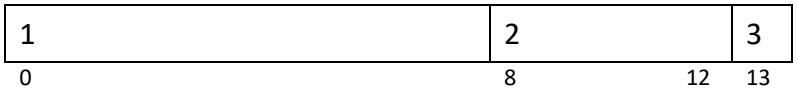
b. SJF algoritmu (bez pretpražnjenja),

c. SJF algoritmu sa vremenom čekanja na procese (*idle time*) t_{idle}=1,

d. SRTF (*shortest remaining time first*) algoritmu, odnosno SJF algoritmu sa pretpražnjenjem.

Kašnjenje dispečera zanemariti.

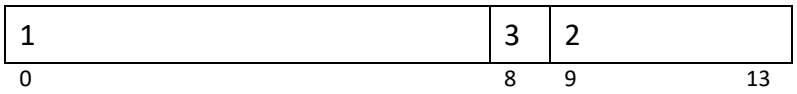
(a) *Gannt*-ova karta dodele procesora - FCFS algoritam:



Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	8	0
P2	12 - 0.4 = 11.6	8 - 0.4 = 7.6
P3	13 - 1 = 12	12 - 1 = 11
sr. vreme:	$\frac{8+11.6+12}{3}$ = 10.53	$\frac{0+7.6+11}{3}$ = 6.2

(b) *Gannt*-ova karta dodele procesora - SJF algoritam:

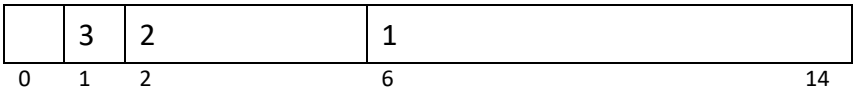


[2] Obratite pažnju: vreme potrebno za kompletiranje procesa se u ovom slučaju računa kao razlika vremena u kom je proces završio sve aktivnosti i vremena nailaska u sistem. Slično važi i za vreme čekanja, koje se računa kao razlika vremena u kom je proces dobio procesor i vrmena nailaska procesa u sistem

Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	8	0
P2	13 - 0.4 = 12.6	9 - 0.4 = 8.6
P3	9 - 1 = 8	8 - 1 = 7
sr. vreme:	$\frac{8 + 12.6 + 8}{3}$ = 9.53	$\frac{0 + 8.6 + 7}{3}$ = 5.2

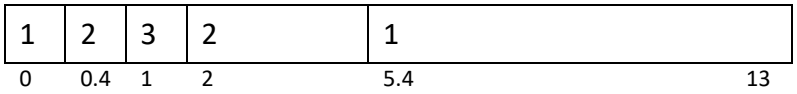
(c) Gantova karta dodele procesora – SJF algoritam sa vremenom čekanja na procese[3] (*idle time*) t_{idle}=1.



Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	14	6
P2	6 - 0.4 = 5.6	2 - 0.4 = 1.6
P3	2 - 1 = 1	0
sr. vreme:	$\frac{14 + 5.6 + 1}{3}$ = 6.86	$\frac{6 + 1.6 + 0}{3}$ = 2.53

(d) *Gannt*-ova karta dodele procesora – SRTF algoritam



Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	13	5.4 - 0.4 = 5
P2	5.4 - 0.4 = 5	2 - 1 = 1
P3	2 - 1 = 1	0
sr. vreme:	$\frac{13 + 5 + 1}{3}$ = 6.33	$\frac{5 + 1 + 0}{3}$ = 2

[3] Vreme čekanja na procese (*idle time*) je vremenski interval u kom planer poslova niskog nivoa (dispečer) čeka da u sistem stigne još nekoliko procesa. Procesima koji u sistem uđu u tom intevalu povećaće se vreme čekanja, ali će se u isto vreme povećati i broj procesa u redu, a samim tim i perfromanse raspoređivanja.

3.11. Dat je sledeći skup procesa čija su vremena nailaska u sistem (*arrival time*) i vremena izvršavanja na procesoru, izražena u milisekundama data u sledećoj tabeli:

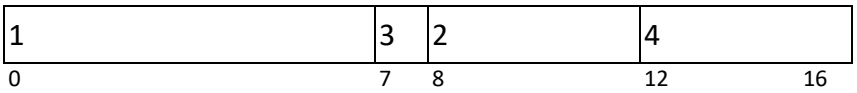
Process	Vreme nailaska	Vreme izvršavanja
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Nacrtati Gantovu kartu i odrediti srednje vreme potrebno za kompletiranje procesa i srednje vreme čekanja ukoliko se raspoređivanje procesa obavlja po:

a. SJF algoritmu.

b. SRTF algoritmu.

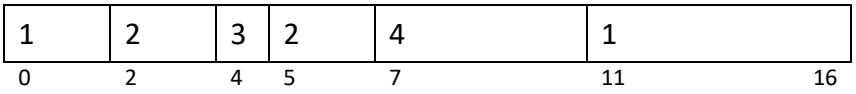
(a) Gantova karta dodele procesora – SJF algoritam:



Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	7	0
P2	12 - 2 = 10	8 - 2 = 6
P3	8 - 4 = 4	7 - 4 = 3
P4	16 - 5 = 7	12 - 5 = 7
sr. vreme:	7	4

(b) Gantova karta dodele procesora – SRTF algoritam:



Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	16	11-2 = 9
P2	7 - 2 = 5	5 - 4 = 1
P3	5 - 4 = 1	0
P4	11 - 5 = 6	7 - 5 = 2
sr. vreme:	7	3

3.12. Četiri procesa su u trenutku t=0 ušli u red čekanja na procesor u sledećem redosledu: P1, P2, P3, P4. Vremena izvršavanja na procesoru za ova četiri procesa su 6, 3, 1 i 7 vremenskih jedinica, respektivno. Ukoliko se raspoređivanje procesa vrši prema Round Robin algoritmu sa kvantumom (a) Q=1, (b) Q=2, (c) Q=3, (d) Q=4, (e) Q=5

- nacrtati *Gannt*-ovu kartu i odrediti srednje vreme izvršavanja na procesoru i srednje vreme čekanja (kašnjenje dispečera zanemariti),
- odrediti koliko puta je obavljena zamena konteksta i koliko je ukupno vremena potrebno da sva četiri procesa završe aktivnosti (kašnjenje dispečera je dl=0.01 vremenskih jedinica).

(a) Gantova karta dodele procesora za slučaj RR, Q=1:

1	2	3	4	1	2	4	1	2	4	1	4	1	4	1	4	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	15	3+2+2+1+1 = 9
P2	9	1+3+2 = 6
P3	3	2
P4	17	3+2+2+1+1+1=10
sr. vreme:	11	6.75

Obavljeno je 16 zamena konteksta, tako da je ukupno vreme potrebno da sva četiri procesa završe aktivnosti: $t = 17 + 16 \cdot 0.01 = 17.16$.

(b) Gantova karta dodele procesora za slučaj RR, Q=2:

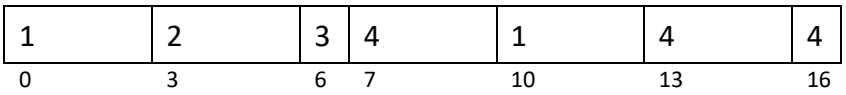
1	2	3	4	1	2	4	1	4	4
0	2	4	5	7	9	10	12	14	16

Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	14	5+3 = 8
P2	10	2+5 = 7
P3	5	4
P4	17	5+3+2=10
sr. vreme:	11.5	7.25

Obavljeno je 9 zamena konteksta, tako da je ukupno vreme potrebno da sva četiri procesa završe aktivnosti: $t = 17 + 9 \cdot 0.01 = 17.09$.

(c) Gantova karta dodele procesora za slučaj RR, Q=3:

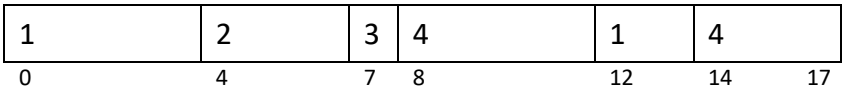


Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	13	7
P2	6	3
P3	7	6
P4	17	7+3=10
sr. vreme:	10.75	6.5

Obavljeno je 6 zamena konteksta, tako da je ukupno vreme potrebno da sva četiri procesa završe aktivnosti: $t = 17 + 6 \cdot 0.01 = 17.06$.

(d) Gantova karta dodele procesora za slučaj RR, Q=4:



Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	14	8
P2	7	4
P3	8	7
P4	17	8+2=10
sr. vreme:	11.50	7.25

Obavljeno je 5 zamena konteksta, tako da je ukupno vreme potrebno da sva četiri procesa završe aktivnosti: $t = 17 + 5 \cdot 0.01 = 17.05$.

(e) Gantova karta dodele procesora za slučaj RR, Q=5:

Vremena potrebna za kompletiranje procesa i vremena čekanja su:

	vreme kompletiranja	**vreme čekanja**
P1	15	9
P2	8	5
P3	9	8
P4	17	9+1=10
sr. vreme:	12.25	8

Obavljeno je 6 zamena konteksta, tako da je ukupno vreme potrebno da sva četiri procesa završe aktivnosti: $t = 17 + 5 \cdot 0.01 = 17.05$.

4.1. a. Šta je to stanje trke (*race condition*)?

b. Od čega zavise vrednosti deljenih podataka nakon završene trke?

c. Da li je krajnji rezultat trke izvestan ili neizvestan?

(a) Stanje trke je situacija u kojoj veći broj procesa konkurentno pristupa zajedničkim podacima.

(b) Krajnje vrednosti zajedničkih promenljivih zavise od toga koji će proces poslednji završiti sa radom, odnonso od sekvence naredbi kojima se ti podaci modifikuju, što dalje zavisi od redosleda prekidnih signala i načina raspređivanja procesa.

(c) U trci se pobednik obično ne zna, tako da je krajnji rezultat neizvestan.

4.2. a. Šta je kritična sekcija?

b. Ko razrešava problem kritične sekcije?

(a) Kritična sekcija je deo programskog koda u kome proces pristupa zajedničkim podacima ili modifikuje zajedničke podatke (kao što su vrednosti memorijskih lokacija, datoteke, itd.)

(b) Programer.

4.3. Koje su osnovne pretpostavke softverske realizacije kritične sekcije?

- Međusobno isključenje,
- proces van kritične sekcije ne sme sprečiti druge procese da uđu u kritičnu sekciju,
- proces ne sme neograničeno dugo da čeka na ulazak u kritičnu sekciju i
- proces ne sme neograničeno dugo da ostane u svojoj kritičnoj sekciji,

4.5. a. Objasnite funkcije ulazne, kritične i izlazne sekcije.

b. Za sledeći kod odrediti naredbe koje pripadaju ulaznoj, izlaznoj i kritičnoj sekciji. Deljiva promenljiva je `share`.

```
while (busy); busy = 1; share++; busy=0;
```

(a) Ulazna sekcija je deo koda u kome proces zahteva od sistema da uđe u kritičnu sekciju,

Kritična sekcija je deo koda u kome proces pristupa ili modifikuje vrednosti zajedničkih podataka,

Izlazna sekcija je deo koda u kome proces obaveštava ostale procese da je napustio svoju kritičnu sekciju.

(b) ulazna sekcija: `while (busy);`

kritična sekcija: `busy=1; share++;`

izlazna sekcija: `busy=0;`

4.6. Objasnite ukratko ideje na kojima su zasnovana značajnija softverska rešenja kritične sekcije: (a) algoritam striktne alternacije, (b) *Dekker-Petersonov* algoritam, (c) pekarski algoritam. Na koliko su procesa ovi algoritmi realno primenljivi?

(a) Samo onaj proces čija je vrednost indeksa jednaka vrednosti zajedničke promenjive za sinhronizaciju (*turn*) može ući u svoju kritičnu sekciju. Algoritam je primenljiv za sinhronizaciju dva procesa.

(b) Binarna promenljiva *flag* se deklariše za svaki proces pojedinačno i ukazuje na to da li proces želi ili ne želi da uđe u svoju kritičnu sekciju. Zajednička promenljiva *turn* ukazuje na proces koji ima prednost ulaska u kritičnu sekciju i obezbeđuje mehanizam međusobnog isključenja. Algoritam je primenljiv za sinhronizaciju dva procesa.

(c) Svakom procesu se dodeljuje broj (procesima se redom dodeljuju sve veći brojevi). Proces ostaje u ulaznoj sekciji sve dok ne postane proces sa najmanjim brojem, odnosno najviim prioritetom, a zatim ulazi u kritičnu sekciju.

4.7. U čemu se ogleda nedeljivost semaforskih operacija *signal* i *wait*?

- Operacije *signal* i *wait* se ne mogu podeliti na više ciklusa.
- Dva procesa ne mogu istovremeno izvršavati ove operacije nad istim semaforom.

4.8. a. Šta znači kada se kaže da je proces "zauzet čekanjem" (*busy waiting*)?

b. Kako se ovaj nedostatak može otkloniti?

(a) Pojava koja se zove *busy waiting* (zaposleno čekanje) nedostatak je softverske i hardverske realizacije kritične oblasti i semafora. Proces ne radi nišra korisno, već u *while* petlji proverava vrednost neke promenljive kako bi saznao da li može ući u svoju kritičnu oblast ili ne (i tako troši procesorsko vreme).

(b) Pojava "zaposlen čekanjem" može se ukloniti uvođenjem proširene definicije semafora. Semafor se definiše kao struktura koju čine vrednost semafora (koja može biti i negativna) i semaforski red (lista pokazivača na procese koji čekaju na semaforu). Proces koji izvršava *wait* operaciju nad semaforom, čija je vrednost 0,

blokira se i prevodi u semaforski red. Proces oslobađa procesor koji se predaje nekom drugom procesu koji nije blokiran.

4.9. a. Kada nepravilno raspoređene operacije signal i *wait* mogu da dovedu do zastoja?

b. Navesti primer za to.

(a) Nepravilno raspoređene operacije signal i *wait* mogu da dovedu do zastoja ukoliko jedan ili više procesa beskonačno dugo čekaju na događaje koji se nikada neće dogoditi.

(b) Neka su S i Q dva binarna semafora čija je inicijalna vrednost 1. Ukoliko se procesi P1 i P2, čiji je kod dole naveden, izvršavaju kvaziparalelno, doći će do zastoja:

```
int P1() {wait(S); wait(Q); .. ; signal(S); signal(Q)}
int P2() {wait(Q); wait(S); .. ; signal(Q); signal(S)}
```

4.10. Sledeći kod sadrži tipičnu programersku grešku u radu sa semaforima.

```
signal(mutex); kritična sekcija; wait(mutex);
```

Šta će se dogoditi u ovom slučaju?

Više procesa će moći da uđe u svoje krtitične sekcije istovremeno. Dalji razvoj situacije i krajnji rezultat su nepredvidivi.

4.11. Sledeći kod sadrži tipičnu programersku grešku u radu sa semaforima.

```
wait(mutex); kritična sekcija; wait(mutex);
```

Šta će se dogoditi u ovom slučaju?

Umesto oslobađanja semafora, nastupiće zastoj.

4.12. Semaforskim tehnikama rešite problem ograničenog bafera.

Za sinhronizaciju se koriste tri semafora:

- brojački semafor `item_available`, kojim se obezbeđuje da potrošač ne može uzeti ništa iz bafera dok proizvođač to ne stavi u bafer. Vrednosti semafora pripadaju intervalu [0,N-1], a inicijalna vrednost je 0;
- brojački semafor `space_available`, kojim se obezbeđuje da potrošač ne može uzeti ništa iz bafera dok proizvođač to ne stavi u bafer. Vrednosti semafora pripadaju intervalu [0,N-1], a inicijalna vrednost je 1;
- binarni semafor `buffer_ready` kojim se bafer štiti kao nedeljivi resurs. Inicijalna vrednost semafora je 1.

Radi jednostavnijeg rešenja, operacije kojima se jedan element stavlja u bafer i uzima iz bafera nazvaćemo `deposit()` i `extract()`.

Kod za proizvođački (*producer*) i potrošački (*consumer*) proces je:

```
int producer() {
  do {
    item next_produced;
    wait (space_available);
    wait (buffer_ready):
      deposit (next_produced);
    signal (buffer_ready);
    signal (item_available);
    } while (1);
}
int consumer() {
  do {
    wait (item_available);
    wait (buffer_ready):
      item next_consumed = extract();
    signal (buffer_ready);
    signal (space_available);
  } while (1);
}
```

4.12. Semaforskim tehnikama rešite problem čitalaca i pisaca.

Za sinhronizaciju se koriste dva binarna semafora:

- `write`, koji obezbeđuje međusobno isključenje procesa pisaca i čitalaca (inicijalna vrednost je 1),
- `counter_ready`, koji štiti promenljivu `read_count`, koja ukazuje na broj aktivnih čitalaca (inicijalna vrednost semafora je 1, a promenljive 0).

Kod procesa pisaca i čitalac je:

```
int writer() {
  do {
    wait (write);
      /* proces menja sadržaj objekta */
    signal (write);
  } while (1);
}
int reader() {
  do {
    wait (counter_ready);
      readcount++;
      if (readcount==1) wait (write);
    signal (counter_ready);
    /* proces čita sadržaj objekta */
    wait (counter_ready);
      readcount--;
      if (readcount==0) signal (write);
    signal (counter_ready)
  } while (1);
}
```

4.13. Semaforskim tehnikama rešite problem večere filozofa. Obezbedite rešenje koje ne izaziva zastoj ni u jednoj situaciji.

Svaki filozof predstavlja proces, a svaka viljuška binarni semafor `fork[i]` čija je inicijalna vrednost 1. Zastoj je moguć ukoliko svi filozofi odjednom uzmu levu viljušku. Zato ćemo dozvoliti da filozof uzme viljuške samo ako su obe slobodne - uzimanje viljušaka proglasićemo kritičnom sekcijom. Kritičnu sekciju štitimo binarnim semaforom `taking_forks` (inicijalna vrednost 0).

Kod za sve filozofe je:

```
int philosopher_i() {
  do {
    wait (taking_forks);
      wait (fork[i]);
      wait (fork[(i+1)%5]);
    signal (taking_forks);
    /* filozof jede */
    wait (taking_forks);
      signal (fork[i]);
      signal (fork[(i+1)%5]);
    signal (taking_forks);
    /* filozof misli */
  } while (1);
}
```

4.14. [Problem uspavanog berberina]. Berbernica se sastoji od čekaonice sa n stolica i radne sobe sa berberskom stolicom. Ukoliko nema mušterija koje zahtevaju uslugu, berberin će zaspati. Ukoliko mušterija uđe u berbernicu i vidi da su sve stolice zauzete, izaći će napolje. Ukoliko je berberin zauzet ali u čekaonici ima slobodnih stolica, mušterija će sesti na jednu od njih i sačekati. Ukoliko berberin spava, mušterija će ga probuditi. Napisati program kojim će aktivnosti berberina i mušterija biti sinhronizovane.

4.15. [Problem pušača]. Na sistemu postoje tri procesa koja predstavljaju pušače i jedan proces koji predstavlja dobavljača. Svaki pušač u beskonačnoj petlji najpre mota cigaretu, a zatim je pali. Da bi pušač smotao cigaretu, potrebni su mu sledeći sastojci: duvan, rizla i šibice. U početnom stanju jedan od pušača ima rizlu, drugi duvan, a treći šibice. Dobavljač ima neograničene zalihe ovih sastojaka, ali im donosi samo po dva sastojka. Pušač koji ima preostali sastojak smotaće cigaretu i signalizirati dobavljaču da je završio cigaretu. Dobavljač će zatim ponovo doneti dva sastojka, čime se ciklus ponavlja. Napisati program kojim će aktivnosti pušača i dobavljača biti sinhronizovane.

5.1. Pod kojim uslovima nastupa zastoj?

Zastoj nastupa ako su istovremeno ispunjena sledeća četiri uslova:

(1) pri korišćenju resursa poštuje se princip međusobnog isključenja,

(2) dodela resursa obavlja se bez pretpražnjenja - procesu se ne može oduzeti resurs,

(3) proces zadržava jedan resurs jer mu je potreban i čeka na resurs koji koristi neki drugi proces,

(4) kružno čekanje (proces P0 čeka na resurs koji drži P1, proces P1 čeka na resurs koji drži P1, ..., proces Pn-1 čeka na resurs koji drži Pn, proces Pn čeka na resurs koji drži P0).

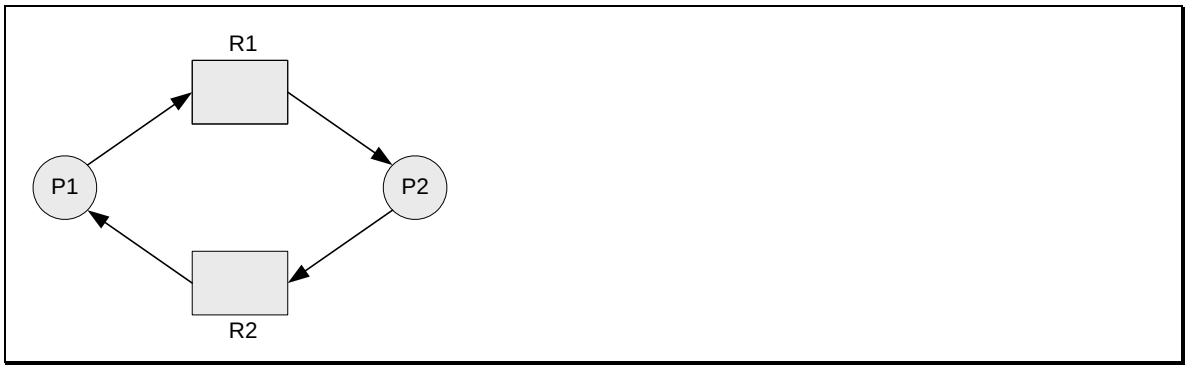
5.2. Da li je moguće da se sistem nađe u stanju zastoja ukoliko na sistemu postoji samo jedan proces? Objasnite.

Nije moguće. Ovo je direktna posledica uslova zadržavanja resursa i čekanja na drugi.

5.3. Sistem čine četiri resursa istog tipa (svaki resurs ima jednu instancu) i tri procesa. Svaki proces zahteva najviše dva resursa. Da li se sistem nalazi u stanju zastoja?

Pretpostavimo da se se sistem nalazi u stanju zastoja. U tom slučaju, svaki proces bi morao da drži jedan resurs i zahteva još jedan koji nije dostupan. Međutim, kako na sistemu ima tri procesa i četiri resursa, jedan proces može da koristi dva resursa istovremeno. Takav proces će završiti posao i osloboditi resurse, a posle toga ostalim procesima mogu biti dodeljena po dva resursa. Sistem nije u stanju zastoja.

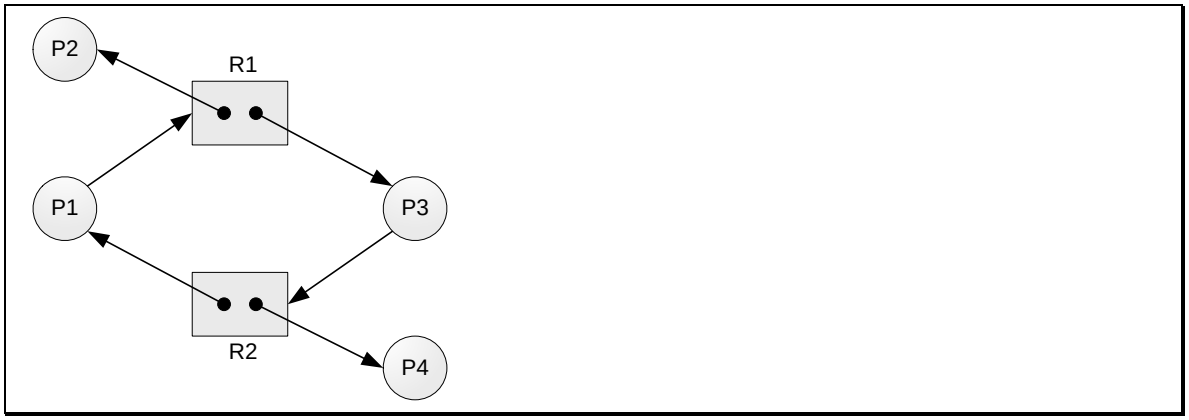
5.4. Da li se sistem opisan grafom dodele resursa sa slike 1. nalazi u stanju zastoja? Obrazložite odgovor.



Slika 1. Graf dodele resursa

Na grafu sa slike 1. postoji kružna putanja: P1-R1-P2-R2. Proces P1 drži resurs R2, a zahteva resurs R1. Proces P2 drži resurs R1, a zahteva resurs R2. Može se zaključiti da se sistem nalazi u stanju zastoja.

5.5. Da li se sistem opisan grafom dodele resursa sa slike 5.3 nalazi u stanju zastoja? Obrazložite odgovor.



Slika 2. Graf dodele resursa

Na grafu sa slike 5.3 postoji kružna putanja: P1-R1-P3-R2-P1, ali zastoja nema jer resursi R1 i R2 imaju po dve instance. Na primer, proces P4 može završiti svoje aktivnosti i osloboditi jednu instancu resursa R2, koja se zatim može dodeliti procesu P3, čime se prekida krug i eliminiše zastoj.

5.6. Posmatrajte sistem u kome se nalaze tri procesa (P0, P1 i P2) i resurs A sa 12 instanci. Stanje sistema je dato sledećom tabelom:

Process	**allocation**	**max**	**Need**
P0	5	10	5
P1	2	4	2
P2	2	9	7

Resurs A ima 3 slobodne instance Odredite da li je sistem u bezbednom stanju.

Sistem je u bezbednom stanju - sekvenca P1, P0, P2 doveće do zadovoljenja potreba svih procesa.

Proces P1 najpre uzima još dve instance resursa, a zatim ih vraća:

Process	**allocation**	**max**	**need**	**available**
P0	5	10	5	
P1	4	4	0	1
P2	2	9	7	

Process	**allocation**	**max**	**need**	**available**
P0	5	10	5	
P1	FINISH			5
P2	2	9	7	

Proces P0, zatim, uzima još pet instanci resursa, a potom ih vraća:

Process	allocation	max	need	available
P0	10	10	0	
P1	FINISH			0
P2	2	9	7	

Process	allocation	max	need	available
P0	FINISH			
P1	FINISH			10
P2	2	9	7	

Na kraju, proces P2 uzima još sedam instanci resursa:

Process	allocation	max	need	available
P0	FINISH			
P1	FINISH			3
P2	9	9	7	

Process	allocation	max	need	available
P0	FINISH			
P1	FINISH			12
P2	FINISH			

5.7. Posmatrajte sistem u kome se nalazi pet procesa (P0, P1, P2, P3 i P4) i tri resursa sa sledećim karakteristikama: resurs A (10 instanci), resurs B (5 instanci), resurs C (7 instanci). Stanje sistema u trenutku t_0 dato je sledećom tabelom:

	Allocation			**Max**		
Proces	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3
P1	2	0	0	3	2	2
P2	3	0	2	9	0	2
P3	2	1	1	2	2	2
P4	0	0	2	4	3	3

a. Da li je sistem u stabilnom stanju?

b. Da li će sistem da odobri zathev P1 [4] (1, 0, 2)?

c. Da li će sistem da odobri zathev P4 (3, 3, 0)?

d. Da li će sistem da odobri zathev P0 (1, 2, 2)?

[4] Proces P1 zahteva dodelu jedne instance resursa A i dve instance resursa C.

e. Da li će sistem da odobri zathev P3 (1, 1, 0)?

(a) Najpre se određuje matrica potreba - need (svaki element matrice need se računa kao razlika odgovarajućih elemenata matrica max i allocation):

	Allocation			**Max**			**Need**			**Available**		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	7	4	3			
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0	3	3	2
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

Zatim se primenom bankarskog algoritma pronalazi sekvenca P1, P3, P4, P2, P0 koja dokazuje da je sistem u stabilnom stanju [5].

(b) Proces P1 izdaje zahtev za dodelom reursa request=(1,0,2). Primenićemo algortiam koji razrešava zahteve za dodelom resursa.

Najpre se proverava da li je request $\leq$ available. Kako je (1,0,2) $\leq$ (3,3,2), uslov je ispunjen, pa se zatim obavlja kvazi-dodela resursa na osnovu zahteva.

	Allocation			**Max**			**Need**			**Available**		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	7	4	3			
P1	3	0	2	3	2	2	0	2	0			
P2	3	0	2	9	0	2	6	0	0	2	3	0
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

Zatim se primenom bankarskog algoritma pronalazi sekvenca P1, P3, P4, P0, P2 koja zadovoljava uslove stabilnosti. To znači da će sistem ispuniti zahtev P1(1,0,2)

(c) Sistem neće odobriti zahtev procesa P4 za dodelu resursa: request=(3,3,0), jer je zahtev veći od raspoloživih resursa.

(d) Sistem neće odobriti zahtev procesa P0 za dodelu resursa: request=(1,2,2), jer se posle obavljene kvazi-dodele, sistem ne nalazi u stabilnom stanju.

[5] Uvek se polazi od procesa sa manjim zahtevima, a kasnije se rešavaju problemi najzahtevnijih procesa..

	Allocation			Max			Need			Available		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	1	3	2	7	5	3	6	2	1			
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0	2	1	0
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

(e) Sistem će odobriti zahtev procesa P4 za dodelu resursa: request=(1,1,0), jer je request $\leq$ available, tj. (1,1,0) $\leq$ (3,3,2), a sistem se posle obavljene kvazi-dodele, nalazi u stabilnom stanju (sekvenca P3, P4, P1, P2, P0).

	Allocation			Max			Need			Available		
Proces	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	7	4	3			
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0	2	2	2
P3	2	1	1	2	2	2	0	1	1			
P4	1	1	2	4	3	3	3	2	1			

5.8. Posmatrajte sistem u kome se nalaze pet procesa (P0, P1, P2, P3 i P4) i četiri resursa. Stanje sistema u trenutku t_0 dato je sledećom tabelom:

	allocation				max			
Process	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2
P1	1	0	0	0	1	7	5	0
P2	1	3	5	4	2	3	5	6
P3	0	6	3	2	0	6	5	2
P4	0	0	1	4	0	6	5	6

U trenutku t_0 slobodna je 1 instanca resursa A, 5 instanci resursa B i 2 instance resursa C, odnosno available=(1,5,2,0). Koristeći bankarski algoritam, odredite:

a. Kako izgleda matrica potreba need?

b. Da li je sistem u bezbednom stanju?

c. Da li će sistem da odobri zathev P1 (0, 4, 2, 0?

(a) Svaki element matrice need se računa kao razlika odgovarajućih elemenata matrica max i allocation:

	allocation				max				need			
Process	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	0	0	0	0
P1	1	0	0	0	1	7	5	0	0	7	5	0
P2	1	3	5	4	2	3	5	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2

(b) Sistem je u bezbednom stanju, jer sledeća sekvenca procesa zadovoljava uslove stabilnosti:

- P0 uzima (0,0,0,0) a zatim vraća (0,0,1,2) $\Rightarrow$ available=(1,5,3,2),
- P3 uzima (0,0,2,0) a zatim vraća (0,6,5,2) $\Rightarrow$ available=(1,11,6,4),
- P2 uzima (1,0,0,2) a zatim vraća (2,3,5,6) $\Rightarrow$ available=(2,14,11,8),
- P1 uzima (0,7,5,0) a zatim vraća (1,7,5,0) $\Rightarrow$ available=(3,14,11,8),
- P4 uzima (0,6,4,2) a zatim vraća (0,6,5,6) $\Rightarrow$ available=(3,14,12,12).

(c) Najpre se proverava da li je request $\leq$ available. Kako je (0,4,2,0) $\leq$ (1,5,2,0), uslov je ispunjen, pa se zatim obavlja kvazi-dodela resursa na osnovu zahteva. Procesu P1 dodeljujemo (0,4,2,0):

	allocation				max				need			
Process	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2	0	0	0	0
P1	1	4	2	0	1	7	5	0	0	3	3	0
P2	1	3	5	4	2	3	5	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2

Posle dodele resursa, slobodna je 1 instanca resursa A i 1 instanca resursa B, odnosno available=(1,1,0,0).

Sistem je u bezbednom stanju, jer sledeća sekvenca procesa zadovoljava uslove stabilnosti:

- P0 uzima (0,0,0,0) a zatim vraća (0,0,1,2) $\Rightarrow$ available=(1,1,1,2),
- P2 uzima (1,0,0,2) a zatim vraća (2,3,5,6) $\Rightarrow$ available=(2,4,6,6),
- P3 uzima (0,0,2,0) a zatim vraća (0,6,5,2) $\Rightarrow$ available=(2,10,9,8),
- P1 uzima (0,3,3,0) a zatim vraća (1,7,5,0) $\Rightarrow$ available=(3,14,11,8),
- P4 uzima (0,6,4,2) a zatim vraća (0,6,5,6) $\Rightarrow$ available=(3,14,12,12).

Sistem odobrava zahtev za dodelom resursa.

5.9. Posmatrajte sistem u kome se nalazi pet procesa (P0, P1, P2, P3 i P4) i tri resursa sa sledećim karakteristikama: resurs A (7 instanci), resurs B (2 instance), resurs C (6 instanci). Stanje sistema u trenutku t_0 dato je sledećom tabelom:

	Allocation			**Request**		
Proces	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	0	0	0
P1	2	0	0	2	0	2
P2	3	0	3	-	-	-
P3	2	1	1	1	0	0
P4	0	0	2	0	0	2

U trenutku t_0 ni jedan resurs nema slobodnih instanci, odnosno available=(0,0,0). Odredite da li je sistem u stanju zastoja, ako:

a. proces P2 zahteva ne zahteva resurse, tj. request(P3)=(0,0,0).

b. proces P2 zahteva jednu instancu resursa C, tj. request(P3)=(0,0,0).

(a) Sistem nije u stanju zastoja. Primenom algoritma za detekciju zastoja može se dokazati da postoji sekvenca P0, P2, P3, P1, P4 nakon koje bi svi procesi završili svoje aktivnosti. Interesantno je da sistem u trenutku t_0 nema raspoloživih resursa, međutim u sistemu postoje dva procesa koji ne traže ni jedan dodatni resurs (procesi P0 i P2). Oni mogu da obave svoje aktivnosti i vrate resurse, koji se zatim mogu dodeliti drugim procesima.

- Kada P0 završi svoje aktivnosti, oslobodiće 1 instancu resursa B. Posle toga je availible=(0,1,0).
- Kada P2 završi svoje aktivnosti, oslobodiće po 3 instance resursa A i C. Nakon toga je availible=(3,1,3).
- Proces P3 je najmanje zahtevan, tako da mu sistem može dodeliti jednu instancu resursa A. Kada P3 završi svoje aktivnosti, on vraća (3,1,1) instanci resursa sistemu, pa je availible=(5,2,4).
- Zatim P1 uzima (2,0,2) i vraća (4,0,2) $\Rightarrow$ available=(7,2,4)
- Na kraju, P4 uzima (0,0,2) i vraća (0,0,4) $\Rightarrow$ available=(7,2,6)

Kada svi procesi završe svoje aktivnosti, na sistemu ostaje 7 instanci resursa A, dve instance resursa B i 6 instanci resursa C.

(b) Sistem je u stanju zastoja. U trenutku t_0, na sistemu nema raspoloživih resursa, međutim proces P0 ne traži ni jedan dodatni resurs. On može da obavi svoje aktivnosti i oslobodi jednu instancu resursa B. Posle toga je availible=(0,1,0). Međutim, dalje ni jedan proces ne može da zadovolji svoje potrebe za resursima.

5.10. Posmatrajte sistem u kome se nalazi pet procesa (P0, P1, P2, P3 i P4) i tri resursa sa sledećim karakteristikama: resurs A (7 instanci), resurs B (2 instance), resurs C (6 instanci). Stanje sistema u trenutku t_0 dato je sledećom tabelom:

	Allocation			Request		
Proces	**A**	**B**	**C**	**A**	**B**	**C**
P0	1	1	0	1	0	0
P1	2	0	0	2	0	2
P2	2	0	2	0	1	0
P3	1	1	1	1	0	0
P4	0	0	2	0	0	2

U trenutku t_0 slobodne su po jedna instanca resursa A i C, odnosno available=(1,0,1). Odredite da li je sistem u stanju zastoja.

Sistem nije u stanju zastoja, jer postoji sekvenca P0, P2, P3, P1, P4 posle koje će svi procesi završiti svoje aktivnosti.

- P0 uzima (1,0,0) a zatim vraća (2,1,0) $\Rightarrow$ available=(2,1,1)
- P2 uzima (0,1,0) a zatim vraća (2,1,2) $\Rightarrow$ available=(4,1,3)
- P3 uzima (1,0,0) a zatim vraća (2,1,1) $\Rightarrow$ available=(5,2,4)
- P1 uzima (2,0,0) a zatim vraća (4,0,2) $\Rightarrow$ available=(7,2,6)
- P4 uzima (0,0,2) a zatim vraća (0,0,4) $\Rightarrow$ available=(7,2,8)